CS211 Lecture: Modeling Dynamic Behaviors of Systems; Interaction Diagrams in UML

last revised September 18, 2007

Objectives:

1. To introduce the notion of dynamic analysis
2. To show how to create and read Sequence Diagrams
3. To show how to create and read Communication Diagrams

Materials:

1. Handout of baseball double play Sequence and Communication Diagrams
2. Ability to project ATM example system
3. Handout of Session Communication diagrams

I. Introduction

A. In trying to understand any system, there are really two aspects of the system to be considered: its static aspects and its dynamic aspects. The static aspects of a system have to do with its component parts and how they are related to one another; the dynamic aspects of a system have to do with how the components interact with one another and/or change state internally over time.

Example: Suppose we were analyzing the game of baseball - in particular, the way that a team functions when it is on defense (out in the field)

1. The static aspects of this system include things like the fact that there are 9 players on the field at a time, and the various positions that they play. In fact, if you were watching a baseball game, this is mostly what you would see - most of the time the players are in their positions, ready for a ball to come their way.

2. The dynamic aspects of this system include the various interactions ("plays") that occur under various circumstances. For example, one such interaction is called a "double play", which can happen when there is a runner on first base and the batter hits a ground ball It commonly takes one of two forms:

   a) If the ball is hit between second and third bases, then the shortstop runs to the ball while the second baseman runs toward second base. The shortstop fields the ball and throws it to the second baseman, who in turn throws it to the first baseman. In baseball jargon, this is called a "6-4-3 double play".

b) On the other hand, if the ball is hit between first and second bases, then the second baseman fields the ball and throws it to the shortstop (covering second) who in turn throws it to first. In baseball jargon, this is called a "4-6-3 double play".

B. Thus far, we have largely discussed static aspects of systems - e.g. a class diagram represents the (static) relationship between the classes comprising a system, etc. CRC cards are developed by thinking through the dynamic behavior of a system, but basically record static information (who is responsible for what; who collaborates with whom, etc.)

C. We will now consider a couple tools that can be used to model the interaction of objects to accomplish a use case. Later, we will look at a tool that can be used to model changes of state of the system as a whole or an individual object in it.

1. Each can be used either for analysis (if our goal is to record how behavior actually transpires in a system we are modeling) or for design (if our goal is to describe a behavior we want to produce.)

2. UML calls these diagrams Interaction Diagrams, of which there are two types:

   a) Sequence Diagrams

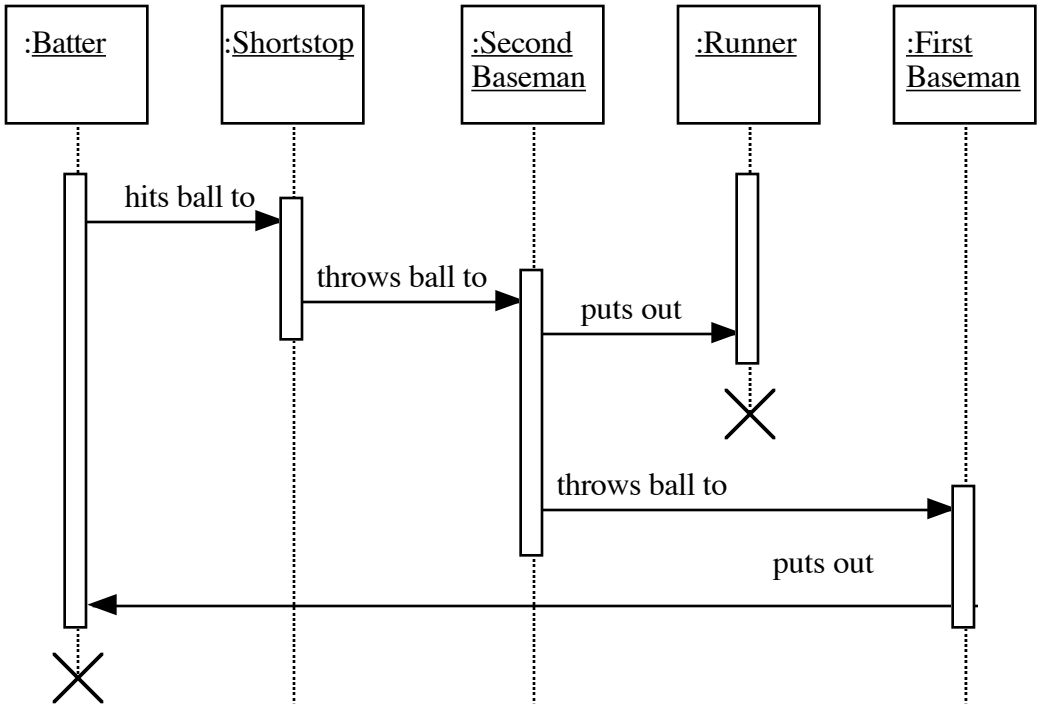
   b) Communication Diagrams (UML 2 name - UML 1 called them colllaboration diagrams.)

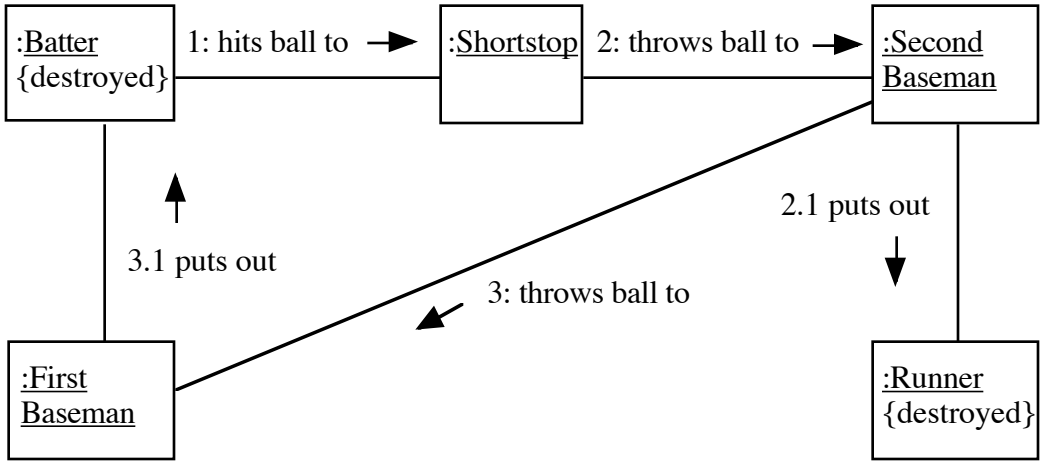
   c) The two types of diagram are equivalent, in the sense that either one can be constructed from information in the other.

3. For example, here are two diagrams that show a "6-4-3" double play: One is a sequence diagram; the other a communication diagram.

   (These are more the kind of diagram that would be produced during analysis than a diagram that would be produced to design software.)

   (HANDOUT)





II. Sequence Diagrams

A. I will focus first on the sequence diagram, because I think it is the easier of the two for a beginner to understand and construct.

PROJECT: Sequence diagram for Session.

B. General observations:

1. An interaction diagram (of either type) depicts a society of objects that work together to carry out a particular use case. Thus, in general, there will be an interaction diagram for each use case.

2. The vertical dimension in a sequence diagram is time. The interaction starts at the top of the diagram, and progresses through time to the bottom.

3. The boxes in the interaction diagram stand for *objects*. Note that UML uses very similar notation for objects and for classes in diagrams - both are represented by rectangles. As you recall from an earlier lecture, two things distinguish the symbol for an object from that for a class:

   a) An object has both an object name and a class name. These are written separated by a colon - e.g. joe: Student. In a UML diagram, it is common for an object to be anonymous - in which case the box contains just the class name, *preceded* by a colon. (If there were a name followed by a colon, it would be an object name of anonymous class.)

   b) As we have already noted, in UML object diagrams, it is common to *underline* the object:class name. (This is not done consistently, though; in fact, I had to add this to the diagram I have handed out, because the drawing tool I used to produce it doesn't provide for this!)

   c) Underneath each object is a dotted line called its *lifeline*. The lifeline indicates when the object is in existence.

      (1) If the lifeline extends the entire length of the diagram, it means that the object exists before the interaction begins and continues to exist after it ends.

         (a) Examples from double-play

            ASK

            Shortstop, Second Baseman, First Baseman

         (b) Examples from ATM Session

            ASK

            CardReader, ATM, CustomerConsole

(2) If the lifeline begins part-way through the interaction, it indicates that the corresponding object is created during the interaction. Likewise, if the lifeline terminates in an "X" before the end of the interaction, this indicates that the corresponding object is destroyed during the interaction.

(a) Examples from double play:

ASK

Batter and Runner are both destroyed in the interaction

(b) Examples from session

Session, Transaction are both created and destroyed in the interaction

(c) It is also possible - though not illustrated in either example - for an object created during an interaction to continue to exist after the interaction completes

(3) Rectangular boxes on the lifeline show times during the interaction when a particular object is *active* - i.e. performing some operation, or waiting for some other object to perform an operation and return a result.

(Discuss examples on charts)

(4) Sometimes, these boxes are shaded, to denote times when the object is actively computing, and left unshaded when the object is waiting for some other object to perform a computation.

Example: Figure 6.19 page 172

4. The diagram shows messages sent from one object to another. We'll illustrate these from the session example. Each message has:

a) A direction - indicated by the arrow near the message description. One of the objects (at the tail of the arrow) is the *sender* of the message and the other (at the head end) is the *receiver* of the message.

b) A name

(1) EXAMPLE: the first message from the CardReader to the ATM is called cardInserted().

(2) *NOTE:* The first message from the ATM object to the Session object is called « create ». This is a stereotype, indicating a special kind of message (actually sent to the class) which results in creating the object. The object does not exist before the « create » message is sent.

The book uses a different notation - the message is sent to the class box. (See, for example, Figure 6.11 on page 160).

This is a case where either notation can be used - though it is good practice to choose one and stick to it. (Sometimes, a particular CASE tool may support just one or the other, of course.)

(3) If the communication is implemented in Java, the message name will generally become the name of a method.

c) A (possibly empty) list of parameters

(1) EXAMPLE: the « create » message from the ATM to Session has as its parameter the atm object (this) that is sending the message.

(2) EXAMPLE: The « create » message from the Session to Transaction has as parameters the ATM object (that created the session in the first place), the Session object (this), the customer's card, and the pin the customer typed.

(3) EXAMPLE: Looking at the messages from and to the CardReader, we see that none have parameters - all parameter lists are empty.

(4) If the collaboration is implemented in Java, the parameters will become actual parameters to the method call.

5. Some messages have a return value. This is indicated by a <u>dotted</u> line going back from the recipient to the original sender.

   EXAMPLE : readCard() returns a card to the Session; readPIN() returns a pin to the session; performTransaction() returns to the Session and indicator as to whether the customer wants to do another.

6. Sometimes a message or group of messages is sent conditionally or repeatedly. This is indicated by enclosing the messages in a box called a <u>combined fragment</u>, with a condition specified.

EXAMPLE: The combined fragment indicating the possibility of multiple transactions.

a) In UML 2.0, there are certain defined ways of labelling a combined fragment, including:

(1) opt (optional) - the fragment is done only when a certain condition is true

(2) alt (alternatives) - two fragments, with one being done in any given case (if .. else)

(3) loop - the fragment is done repeatedly while a certain condition is true or for certain values

(4) (There are others which we will not touch on)

NOTE: The ATM Example was done using UML 1, and so does not have labels on the fragments

b) If the interaction is implemented in Java, a fragment will be implemented by an if, if .. else, while or for statement, with the condition on the fragment becoming the condition for the Java statement.

7. Not shown in the example is the possibility than an object might send a message to itself. We will see an example of this in a moment.

C. SHOW ADDITIONAL SEQUENCE DIAGRAMS ON THE WEB - note messages sent by a Transaction object to itself (to take advantage of polymorphic methods)

III. Communication Diagrams

A. Another form of interaction diagram in UML is the Communication Diagram. (Called a Collaboration Diagram in UML 1)

NOTE two diagrams in Double-Play handout

HANDOUT: Communication Diagram for Session Use Case (not included in example on the web)

1. Comparing the two types of diagram for the same use case is instructive.

a) The sequence diagram focuses on *time sequence*

b) The communication diagram focuses on *object relationships*

2. In a communication diagram, as in a sequence diagram, each object participating in an interaction is represented by a box.

3. The objects are connected by lines representing links. If a link has no arrows on it, it is taken as being bidirectional: the object at each end knows about the object at the other end. If there is an arrow, it is unidirectional: the object at the tail end knows about the object at the head end, but not vice versa. In the case of a bidirectional link, we must indicate the direction of the message with an arrow, as shown on the diagram.

   EXAMPLE: Who knows about whom in the example diagram?

4. Time sequence is specified by the use of *sequence numbers*. a number followed by a colon (e.g. 1:. 3.2: etc.)

   a) The sequence numbers indicate the order in which the messages are sent. Message 1 is sent first, then 2, etc.

   b) Multiple numbers separated by dots represent *nesting* of messages.

      EXAMPLE: carrying out what is required by message 3 (performSession()) results in the sending of message 3.1 (readCard()) then 3.2 (getPIN()), etc.

   c) NOTE: To avoid excessive complexity, it is possible to draw layers of communication diagrams. A coarse grained diagram may show only the main level of messages (1, 2, 3). These may be refined in subsequent diagrams - e.g. there might be a separate diagram showing the messages associated with top-level message 2 (2.1, 2.2, etc.). We will not deal with examples complex enough to call for this.

5. If a message returns a value to its caller, this is denoted by an assignment operator - so that the message name looks like a function call. (The := operator is used for assignment in Pascal and Ada, and has been adopted by UML instead of =.) (Sometimes this is also used in sequence diagrams for messages that return a result immediately)

B. As a general rule, the implementation of each use case will be described by an interaction diagram (of one type or the other) showing the objects that are needed to carry out that use case.

   1. Thus, there will be at least one interaction diagram for each use case.

      EXAMPLE: Show links from Use Case page to Interactions on the web example

2. Interaction diagrams can be created at the same time as CRC cards, to detail the actual flow of messages that we discover doing the CRC role-playing. Or, they can be created later, during detailed design of the various classes.

3. It is also possible to use interaction diagrams for other purposes - e.g. if a class has an operation (method) that is sufficiently complex, that operation may be designed using its own interaction diagram.

C. CLASS EXERCISE: Do Exercise 6.8 (page 179; answers p. 384)

D. If time, do 6.9